Esercitazione

Laboratorio Sistemi Operativi

Aniello Castiglione
Email: aniello.castiglione@uniparthenope.it

Esercizio 1

- Si realizzi un programma in C e Posix sotto Linux che, con l'ausilio della libreria Pthread
  - lancia n thread per calcolare la somma degli elementi di ciascuna riga di una matrice nxn di interi generati casualmente in un intervallo [0,255], allocata dinamicamente
  - Il calcolo della somma degli elementi di ciascuna riga deve essere effettuato concorrentemente su tutte le righe, secondo la seguente modalità:
    - il thread i-esimo con i pari, calcola la somma degli elementi di indice pari della riga i-esima
    - il thread con indice i dispari, calcola la somma degli elementi di indice dispari della riga i-esima.
  - Calcolate le somme parziali, si provvederà a ricercarne il minimo ed a stamparlo a video.

Sol 1

```
void *SommaP(void*); /*operazione che effettua il thread*/
/*
mutex:semaforo;
minimo:minimo somme parziali*/
struct
{
   pthread_mutex_t mutex;
   int minimo;
}shared={PTHREAD_MUTEX_INITIALIZER,-1};
/* matrice condivisa tra i thread*/
   int **mat;
/*numero righe colonne e righe informazione che devono sapere tutti i
thread*/
   int n;
int main(int argc,char**argv)
{
   int i,j;
   pthread_t *t;
/*se non viene inserito un dato in ingresso esce*/
   if(argc!=2)
   {
     printf("errore dati in ingresso");
     exit(1);
   }
   n=atoi(argv[1]);
   int *tids[n]
/*allocazione dinamica matrice*/
   mat=(int**)malloc(n*sizeof(int*));
   for (i=0;i<n;i++) mat[i]=(int*)malloc(n*sizeof(int));
```

```
/*inserimento dei dati nella matrice casualmente*/
   for(i=0;i<n;i++)
   {
     for(j=0;j<n;j++)
     {
        mat[i][j]=rand()%256;
        printf("%d ",mat[i][j]);
     }
     printf("\n");

   }

/*creazione dinamica del vettore di thread*/
   t=(pthread_t*)malloc(n*sizeof(pthread_t));
   for(i=0;i<n;i++)
   {
           tids[i]=(int*)malloc(sizeof(int));
           *tids[i]=i;
           pthread_create(&t[i],NULL,SommaP,(void*)tids[i]);

   }
/*attesa fine thread*/
   for(i=0;i<n;i++)
   {
     pthread_join(t[i],NULL);

   }
/*stampa minimo delle somme parziali*/
   printf("il minimo delle smme parziali e' %d\n",shared.minimo);
   exit(0);
}
```

```
void *SommaP(void* in)
{
   int riga,j,somma=0;
/*riga su quale effettuare somma parziale*/
   riga=(int)in;

/*se il valore e' pari somma le porzione pari altrimenti le
dispari*/
   if ((riga%2)==0)
         for(j=0;j<n;j=j+2) somma=somma+mat[riga][j];
   else
         for(j=1;j<n;j=j+2) somma=somma+mat[riga][j];
  /*stampa la sua somma parziale*/
  printf("sono thread %d ,somma parziale %d\n",riga,somma);
/*accesso esclusivo sulla variabile minimo*/
  pthread_mutex_lock(&shared.mutex);
```

```
/*verifica se la somma e minone del
minmo*/
    if(shared.minimo>somma)
          shared.minimo=somma;
  /*rilascia il semaforo*/
  pthread_mutex_unlock(&shared.mutex);
/*fine thread*/
  pthread_exit(0);
}
```

Esercizio 2

- Si realizzi un programma in C e Posix sotto Linux che, utilizzando la libreria Pthread
  - lancia n thread per cercare un elemento in una matrice nxn di interi
  - Ognuno dei thread cerca l'elemento in una delle righe della matrice
  - Non appena un thread ha trovato l'elemento cercato, rende note agli altri thread le coordinate dell'elemento e tutti i thread terminano (sono cancellati)

Sol 2

```
int coord[2]={-1,-1};
int key,righe,colonne;
pthread_mutex_t coord_mutex;
int **matrice;
pthread_t *threads;
void crea_matrice();
void stampa_matrice();

void *ricerca(void *tid)
{
  int k,j, *mytid;

  mytid = (int *) tid;

  printf ("Thread %d doing search %d \n",*mytid,*mytid);
  for (j=0; j < colonne; j++) {
     if (matrice[*mytid][j] == key){
        pthread_mutex_lock(&coord_mutex);
        coord[0]=*mytid;
        coord[1]=j;
     for (k=0;k<righe && k!=*mytid;k++){
                printf("%d %d\n",k,*mytid);
                pthread_cancel(threads[k]);
        }
        pthread_mutex_unlock(&coord_mutex);
        pthread_exit(NULL);
     }
  }

  pthread_exit(NULL);
}
```

Sol 2

```
int main(int argc, char**argv){
  int *tids, i;
  righe = atoi(argv[1]);
  colonne = atoi(argv[2]);
  crea_matrice(righe,colonne);
  stampa_matrice();
  printf("Elemento da ricercare: ");
  scanf("%d",&key);
  threads = calloc(righe,sizeof(int));
  tids = calloc(righe, sizeof(int));

for (i=0; i<righe; i++) {
    tids[i] = i;
    pthread_create(&threads[i], NULL, ricerca, (void *) &tids[i]);
    }
for (i=0; i<righe; i++) {
    pthread_join(threads[i], NULL);
  }
  printf ("Fatto. Coordinate = %d %d\n", coord[0],coord[1]);
  exit(0);
}
```

Sol 2

```
void crea_matrice(){
        int i,j;
        matrice = (int **)calloc(righe,sizeof(int *));
        for(i=0;i<righe;i++)
                matrice[i] = (int *)calloc(righe,sizeof(int));
        for(i=0;i<righe;i++)
                for(j=0;j<colonne;j++)
                        matrice[i][j] = rand() % 20;
}

void stampa_matrice(){
int i,j;
for(i=0;i<righe;i++){
                for(j=0;j<colonne;j++)
                        printf("%d   ",matrice[i][j]);
                printf("\n");
}
}
```

Esercizio 3

Facendo uso della libreria Pthread, realizzare un programma in cui un thread scrittore, dato un intero N da riga di comando (dove 10<N<=15), scrive in un file nella prima posizione, uno alla volta ed ogni ½ secondo, la sequenza di Fibonacci di ordine N, alternandosi con un thread lettore che legge, uno alla volta dalla prima posizione del file, i numeri scritti dal thread scrittore. Un terzo thread attende la lettura dell' N-esimo intero, quindi stampa a video il messaggio "Operazioni concluse, arrivederci dal thread: tid", attende 5 secondi e termina.

Esercizio 3 Sol.

```
sem_t *sem1,*sem2;
int n;
struct timespec attesa;
attesa.tv_sec = 0;
attesa.tv_nsec = 500000000L;
void *scrittore(void *argc);
void *lettore(void *argc);
void *stampa(void *argc);
int fd;
pthread_mutex_t mutex;
pthread_cond_t cond;
int cnt=-1;
int main(int argc, char **argv){

        if ( argc == 2 ){
          n = atoi(argv[1]);
        }
        sem_unlink("/sem1"); sem_unlink("/sem2");
       sem1 = sem_open("/sem1",O_CREAT,0777,1);
       sem2 = sem_open("/sem2",O_CREAT,0777,0);
```

Esercizio 3 Sol.

```
if ( n < 10 || n >= 15 ){
                printf("Errore\n"); return 0;
        }

        pthread_t thread1,thread2,thread3;



        fd = open("numeri",O_RDWR|O_CREAT|O_TRUNC,0777);


        pthread_create(&thread1,NULL,scrittore,NULL);
        pthread_create(&thread2,NULL,lettore,NULL);
        pthread_create(&thread3,NULL,stampa,NULL);

        pthread_join(thread1,NULL);
        pthread_join(thread2,NULL);
        pthread_join(thread3,NULL);


        return 0;
}
```

Esercizio 3 Sol.

```
void *scrittore(void *argc){
        int i;
        int a; int b;
        int f;
        b = 1; a = 0;
        for(i=0; i<=n; i++){
           sem_wait(sem1);
                f = b+a;
                a = b;
                b = f;
                lseek(fd,0,SEEK_SET);
                write(fd,&b,sizeof(int));
                printf("Ho scritto: %d\n",b);
                nanosleep(&attesa,NULL);

           sem_post(sem2);
        }

}
```

Esercizio 3 Sol.

```
void *lettore(void *argc){
      int i;
      int buff;
      while(1){
         sem_wait(sem2);
         lseek(fd,0,SEEK_SET);
         read(fd,&buff,sizeof(int));
         printf("Ho letto: %d\n",buff);
         sem_post(sem1);
         pthread_mutex_lock(&mutex);
         cnt++;
         if ( cnt == n ){
             pthread_cond_signal(&cond);
             pthread_mutex_unlock(&mutex);
             return;
         }
         pthread_mutex_unlock(&mutex);
}
```

Esercizio 3 Sol.

```
void *stampa(void *argc){
     pthread_mutex_lock(&mutex);
     while ( cnt < n){
          pthread_cond_wait(&cond,&mutex);
     }
     pthread_mutex_unlock(&mutex);
     pthread_t tid = pthread_self();
     printf("Operazioni concluse,
arrivederci dal thread:
%u\n",(unsigned)tid);
}
```